ESD-TR-75-62

MTR-2924

DOWNGRADING IN A SECURE MULTILEVEL COMPUTER SYSTEM: THE FORMULARY CONCEPT

D. F. Stork

MAY 1975

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
Hanscom Air Force Base, Bedford, Massachusetts

Project No. 7070
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract No. F19628-73-C-0001

REVIEW AND APPROVAL

This technical report has been reviewed and is approved for publication.

William R Price

WILLIAM R. PRICE, 1Lt, USAF
Project Engineer

Roger R. Schell

ROGER R. SCHELL, Major, USAF
Project Engineer

FOR THE COMMANDER

Robert W O'Keefe

ROBERT W. O'KEEFE, Colonel, USAF
Director, Information Systems
Technology Applications Office
Deputy for Command & Management Systems

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ESD-TR-75-62	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DOWNGRADING IN A SECURE MULTILEVEL COMPUTER SYSTEM: THE FORMULARY CONCEPT		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER MTR-2924
7. AUTHOR(s) D. F. Stork		8. CONTRACT OR GRANT NUMBER(s) F19628-73-C-0001
9. PERFORMING ORGANIZATION NAME AND ADDRESS The MITRE Corporation Box 208 Bedford, MA, 01730		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Project No. 7070
11. CONTROLLING OFFICE NAME AND ADDRESS Deputy for Command and Management Systems Electronic Systems Division, AFSC Hanscom Air Force Base, Bedford, MA, 01731		12. REPORT DATE MAY 1975
		13. NUMBER OF PAGES 60
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

ACCESS CONTROL
COMPUTER SECURITY
DOWNGRADING
MATHEMATICAL MODELS

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The tasks to be performed during the development of the Secure Multilevel Data Base System include the construction of a capability for the transformation of data of higher levels of classification to data at lower levels. This capability is to be part of a system in which access control is based upon a security kernel for the PDP-11/45. In this report a mechanism for facilitating downward transformations is developed, and the impact of the mechanism upon both the Bell-LaPadula model of secure computer

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. ABSTRACT (Concluded)

systems and the security kernel is discussed. An expository treatment of the model and kernel is also included.

TABLE OF CONTENTS

TABLE OF CONTENTS (Concluded)

LIST OF ILLUSTRATIONS

SECTION I

INTRODUCTION

PURPOSE OF THE PAPER

This paper will explore the possibility of incorporating a facility for downgrading (by which we mean the transformation of information resident in a data set of a given security level into information which may reside in a data set of a lower level) in a secure multilevel data base system in which access control is in accordance with a mathematical model of secure computer systems. The downgrading facility will enable the data base system to support a sensor correlation environment.

Within the paper, prior to the introduction of the *formulary mechanism* that permits the downgrading capability, an exposition is presented of the mathematical model and the security kernel upon which the formulary concept and formulary mechanism are superimposed. Then the concept and mechanism are developed and their impact upon the model and kernel is assessed.

In the remainder of this introductory section we will establish the context of the ensuing discussion and outline the organization of the paper.

THE CONTEXT

One of the many topics under the heading of computer security is that of access control. This topic, which has been pursued independently of others such as privacy policies and physical security, provides our context.

The work done at MITRE in the area of access control centers on the concept of a reference monitor, which was identified by the ESD Computer Security Technology Planning Study Panel. [1]. A reference monitor is a hardware/software mechanism that mediates all attempts by subjects ("accessors") to gain access to objects ("accessees") within a computer system. The mediation requires consultation of a data base that describes the security state of the system. The reference monitor satisfies three postulates: (1) it is invoked on every access attempt; (2) it is tamperproof; (3) it is small and certifiably correct.

A mathematical model was constructed [2] that (1) identifies in abstract form a data base that describes the security state of a computer system; (2) defines security in terms of relations among elements of that data base; and (3) provides rules for dynamic alteration of that data base in a manner that preserves security and some other desirable properties of the system.

A security kernel (i.e., software portion of the reference monitor) was designed for a PDP-11/45 equipped with a memory management unit. The role of the security kernel is the maintenance of a security data base and the physical resources of the machine in response to requests by processes for access to objects and for alterations in the security data base. The design of the security kernel is based upon the mathematical model of secure systems. [3]

Among the goals of Project 7070 is the demonstration of the PDP-11/45 security kernel in support of a secure multilevel data base system. Some processes using this system will have the ability to downgrade data within the system. The existence of such an ability raises several questions about the underpinnings of the data base system:

Can the mathematical mode accommodate downgrading?
Can the model be altered to do so?
Is the security kernel hospitable to downgrading?
What modifications are indicated in the kernel in order to admit downgrading?
What aspects of downgrading are apparent to a user?

It is to these questions that this paper is addressed.

ORGANIZATION OF THE PAPER

Section 2 provides brief accounts of the Bell-LaPadula model of secure systems and of the security kernel designed for a PDP-11/45 that is based upon that model.

Section 3 explores the meaning of "downgrading" and associated terms, defines and evaluates two approaches (on the level of mathematical model) to downgrading, and decides to use the approach which exploits the "trustworthy subject" concept of the model.

In Section 4, the mechanisms of a formulary (a process which does downgrading) are discussed within the context of the security kernel. Several design choices are described, and the effect upon these choices of controls already existing in the kernel are discussed. Kernel modifications attendant upon incorporation of a formulary mechanism are also explored, as is the problem of object integrity.

Section 5 deals with a model of data paths, which are relevant to the issue of object integrity. The model builds upon the Bell-LaPadula model.

SECTION II

THE MATHEMATICAL MODEL AND THE SECURITY KERNEL

A BRIEF ACCOUNT OF THE BELL-LaPADULA MODEL

In [2], Bell and LaPadula formalized concepts of computer security in a mathematical model of a secure computer system. The elements of this model include

objects
subjects
modes of access and access permissions
security level and formal access categories
object hierarchy
record of current access
access rules

Objects are entities within a computer system to which access must be gained in the course of the system's use.

Subjects are those entities that seek to gain access to objects.

The basic modes of access which a subject may enjoy with respect to an object are the read, write, append, and execute modes.* Access permissions for particular modes are recorded on a per-subject, per-object basis in an access matrix.

Each object is assigned a security classification and formal access category, presumably reflecting the level of sensitivity of

*As used here, "write" includes "read", while "append" is a "pure" write.

the object. Each subject has maximum and current security classifications and formal access categories (i.e., four security descriptors per subject). The set of security classifications is linearly (i.e., sequentially) ordered, while the collection of access categories is ordered by inclusion.

The _object hierarchy_ can be thought of as a directed tree whose set of nodes consists of objects in the system; this tree has a single root node. Note that every object in the hierarchy (except the root object) has a single parent object. The hierarchy is _compatible_ with the object classification and category labels if the classification and category of each object are at least as high as those of its parent.

The _record of current access_ is, as its name implies, a record of which subjects are currently accessing which objects in which modes. To clarify the distinction between current access and access permission, note that the former has an immediate quality (it is what is happening now) while the latter is indefinite (it describes what may be allowed to happen at some time).

The _system state_ consists of the access matrix, the classification and category labels of subjects and objects, the object hierarchy, and the record of current access. It contains all security-related information about a computer system at a given time. A state

(i) is _secure_ if no subject is accessing (in read or write mode) an object whose classification or category is greater than the maximum classification or category of that subject;

(ii) _satisfies the *-property_ if a subject may only (a) write an object at its current level; (b) read an object of its current level or a lower level; (c) append to an object of its current level or a higher level. The model also includes a provision for _trustworthy_ subjects, which are those which are exempt from *-property restrictions.

(iii) is compatible if the object hierarchy is compatible with the classification and category labels.

Within the context of the security model, a computer system is a collection of state sequences. The progression of a system from state to state is governed by the application of access rules. An access rule is a function whose arguments are of the form (request, state) and whose values are of the form (decision, state). A request is made by a subject in order to:

gain or release current access to a specified object in a specified mode;
create a specified object at a specified place in the object hierarchy;
delete a specified object from the system;
give or rescind access permissions for a specified object in a specified mode of access to a specified subject;
change the current classification and category of the requesting subject.

A decision to grant or deny a request is made and the system state is modified or left unaltered according to the algorithm of the particular rule invoked. The access rules do not enforce security, *-property, and compatibility directly; rather, they preserve these properties if the properties are possessed by the input state. Thus, if the initial state of the system is secure, satisfies the *-property, and is compatible, then the system will have these qualities throughout its history.

An informal interpretation of the elements described above will provide a link between the model and reality. One may think of objects as files, and of subjects as users or user surrogates (processes) within the system. (This is not an exhaustive list of examples. Objects could be data or program files, input/output devices, messages which are sent to processes -- in short, anything which can be accessed is an object.)

The notions of security and *-property are intended to prevent compromise of information in the sense of unauthorized disclosure; an untrustworthy subject cannot gain a combination of access rights that will enable it to read a high-level object and write one of a lower level. The object hierarchy models a collection of directory objects and non-directory objects which can be thought of as an abstraction of a hierarchical file system.* The access rules provide a mechanism by which the objects can change in number and in their relations with subjects. The security level functions, the access permissions, the record of current access, and the object hierarchy all provide criteria for the application of the access rules and reflect the consequences of such applications.

---

*Enforcement of compatibility prevents at least two undesirable situations from occurring. The first situation affects security. If an unclassified object were attached to (i.e., described in) a secret directory, a secret process could not write directly in the unclassified object. However, if the process enjoyed write access to the directory, it could write and alter the attributes of the object, thereby affecting the accessibility of the object to lower-level processes. Degrees of accessibility (as observed by a lower-level process) could serve as a signaling alphabet and result in the implementation of a write-down path in violation of the intent of the *-property.

The second consideration is the difficulty - disclosure problems aside - of using a system in which a process cannot always read the directories of objects it needs to use.

Other researchers in computer security have worked on an alternative model addressed to the problem of sabotage as well as models concerned with compromise (unauthorized disclosure). The sabotage model deals with unauthorized upward-directed modification of objects rather than unauthorized downward-directed disclosure. The sabotage model is also phrased in terms of subjects, objects, and security levels, but the levels measure degrees of trust rather than degrees of privilege. (For example, an unclassified compiler written by a person with a secret clearance might be trusted -- i.e., useable -- to compile secret code.) In order to prevent sabotage, a subject may only read objects of levels equal to or above its own level, may only write objects of levels equal to or below its own level. The concept of levels of trust appears later in this paper in the discussion of object integrity under the name of levels of safety.

THE PDP-11/45 SECURITY KERNEL

This section contains an account of how the Bell-LaPadula model has been interpreted for the design of a security kernel for a PDP-11/45 equipped with Memory Management Unit.

Segment Objects

The Memory Management Unit (MMU) option of the PDP-11/45 permits the identification and protection of one type of object, which is called a segment. [4] From the point of view of the MMU, a segment is a region in main memory consisting of contiguous locations and is of specified starting address and length. Every memory reference by the processor is routed through the MMU, where (1) the address is translated from a virtual to a physical address, and (2) protection codes pertaining to the segment in which the location addressed lies are checked and enforced. The hardware supports enforcement of read, read/write, and no-access protection, where enforcement may take the form of traps or aborts upon detection of attempted access.

The access rules of the model are algorithmic in form and lend themselves to implementation as procedures. When these procedures are coded and positioned so that they may be applied for the purpose of access control (i.e., placed in the main memory of the computer), they are objects and need to be protected. The PDP-11/45 meets this need for protection by providing a hierarchy of three machine states (or modes, or domains) called the kernel, supervisor, and user domains. The hardware effects the hierarchical ordering of domains by:

1) permitting the execution of certain machine instructions in the kernel domain only, and

2) restricting the manner in which the instructions which pass control from domain to domain may operate.

The security kernel protects itself by

1) ensuring that its own procedures are the only ones that execute in kernel mode, and

2) executing interpretively all attempts to access the security kernel data base that originate with processes executing outside the kernel.

Interpretive execution of access attempts within the kernel permits objects accessed in the kernel domain to be portions of segments, whereas a directly accessed object outside the kernel must be coextensive with a single segment.

With the MMU in operation, a maximum of sixteen segments is concurrently accessible within each domain. This maximum is achievable if a distinction is made between instruction (read/execute or read/write/execute) segments and data (read or read/write only) segments, in which case eight segments of each type are available. If it is not deemed desirable or feasible to make the distinction between data and instructions, then only eight segments (per domain) of a single undifferentiated type will be available. (Such is the case for the current security kernel design.) Accommodation of a larger number of segments within main memory requires management of the segmentation registers. In order to satisfy a need for a still larger number of segments, swapping between secondary storage and main memory will be necessary. With swapping in use, it is no longer viable to identify a segment as a region of main memory. Rather, the definition of a segment (at least, one which is not permanently resident in main memory) must be altered, to wit: A segment is a collection of contiguous virtual memory locations which is identified by its "home" disk address, and which may be inserted into an appropriately sized and protected region of main memory.

Processes

The type of subject recognized by the security kernel is an ordered pair whose components are a process and a domain. [5] A process, in turn, is identified by the kernel as operating on behalf of a user/project pair. Information that describes processes is contained in data structures called process segments and the process table that are accessed in the kernel domain.

The word "domain" has a dual meaning. One aspect refers to the machine state. In the second meaning, a domain is the environment of programs and data in which a process is operating. A domain (in the machine sense) allows construction of an address space consisting of up to eight concurrently accessible segments, and therefore facilitates the definition of a domain in the environmental sense.

The Security Data Base and the Segment Hierarchy

The security data base in the mathematical model consists of an access matrix (which describes access permissions, or need-to-know), classification and category functions, a record of current access, and a description of the object hierarchy. In the current security kernel design, this data is not centralized and monolithic, but is distributed through the system.

The access matrix is stored column-wise by segment in the directory of a segment object. The directories are themselves segment objects, and as such are subject to access controls. Access controls on directory segments are more restrictive than those on non-directories in that directories can only be accessed in the kernel domain; i.e., interpretively.

Classification and categories for a segment object are maintained in both a segment's directory and in the record of current access (whose interpretation will be described shortly). A subject's classification and categories are recorded in the process table and a process segment.

The record of current access, which is called the Active Segment Table, is resident in main memory and is accessed in the kernel domain. It contains physical (i.e., implementation) details about segments as well as information corresponding to elements of the model.

Finally, the structure of the object hierarchy is described locally by pointers in the directories. The "parent" of the model corresponds to the "directory" of the kernel.

The security and access attributes of segment objects and processes executing in non-kernel domains are recorded explicitly in data structures accessed in the kernel domain. However, the attributes of objects (other than directories) accessed in the kernel domain are implicitly described by the operation of the security kernel.

For example, every process may execute kernel procedures without security checking; hence, every process has read access permission to the security kernel procedures, and the security level of these procedures is the low for the system. Furthermore, treatment of the seurity kernel as a privileged section limits current access to the kernel to the process identified as <u>the</u> current process. This mechanism implies a single-process limit on entries in the record of current access in which the object component is a kernel procedure. Implicit contributions of this last-mentioned type to the security state information are created and destroyed by operations on the kernel semaphore (p's and v's, respectively).

Further elucidation of the topic of implicit security state information awaits development of the security kernel validation effort.

<u>Access Rules - The Kernel Functions</u>

The security kernel includes over thirty procedures, approximately half of which are callable by processes that are not operating in the kernel domain. The functions that are not externally callable are called, either directly or indirectly, by functions that are callable; the noncallable functions are invisible outside the security kernel. The noncallable functions deal with management of the computer's physical resources and will not be discussed further here.

Of the externally callable functions, some correspond to access rules of the mathematical model; their names are: give, rescind, create, delete, getr, getw, enable, disable, dconnect. The functions give, rescind, create, delete, getr, getw correspond directly to the access rules of the same or similar names in the model, and dconnect corresponds to "release" of the model. The kernel contains no separate get-execute or get-append rules. Also, there is no change-security-level rule in the kernel since the distinction made by the model between maximum and current security levels has not been implemented.

The functions getr and getw are invoked, respectively, when read/execute and write/read/execute access is desired, and their invocation results in entries to the Active Segment Table. In order to make use of access privileges for a given segment, the segment must be in main memory, and segmentation registers must be loaded appropriately; these chores are accomplished by invocation of the function enable. The function disable undoes enable. Since enable and disable are externally callable, responsibility for management of a process' address space can be placed outside the security kernel.

The callable functions startp and stopp deal with process creation and destruction, respectively. Although these functions do not correspond directly to any rules of the mathematical model, they are logically dual to the functions create (-object) and delete (-object). Due to this duality, they fill a need which the model did not address, but without departing from the spirit of the model.

Two more callable functions which do not correspond directly to rules of the model but are nonetheless in accord with its principles are ipcsend and ipcrcv, which implement an interprocess communication facility. Ipcsend effects a pure write (append) of a message from the requesting process to a designated receiving process. The message is considered to be of the same security level as the sending process; consequently, the level of the receiving process must be at least as high as that of the sending process, as is appropriate for read access. The security checking is done within _ipcsend_ and is accordingly omitted from _ipcrcv_.

Startp may only be invoked by one particular trustworthy process, called the executive process. Its security level is high for the system, so that it may receive messages from any process. Trustworthiness allows the executive process to send messages to any process.

The functions _p_ and _v_ are operations on semaphores and are used for control of multiple accesses of processes to segments and for synchronization signals between processes. These functions are also used to maintain the security kernel as a critical section. That is, at most one process can access the security kernel at one time; this restriction corresponds to the strict sequentiality in the processing of subject requests that is employed in the mathematical model.

SECTION III

DOWNGRADING IN THE CONTEXT OF THE BELL-LaPADULA MODEL

INTRODUCTION

Definition of Downgrading

In the context of the model downgrading refers to the construction or use of an information path from a given object to one that has either a lower classification or whose categories form a proper subset of the categories of the given object. Such action can take any one of several forms.

Downgrading Approaches

One method of downgrading consists of overriding the current security level description of an object and writing a new description for that object. This method of downgrading will be referred to as "level-change". Declassification of paper documents by level-change is done on a prescribed schedule as described in [6].

Writing information with a relatively high security level into an object of a lower level is another way of downgrading information. This method will be referred to as "write-down".

Sanitization

"Sanitization" is a third downgrading concept, but it is not wholly distinct from the two given above. It consists of transforming information in such a way that the resultant is less sensitive than the original. It is not distinct from level-change and write-down for two reasons.

First, sanitization can be superimposed on either form of downgrading discussed above. With the level-change method, sanitization is a single-level procedure that precedes the actual downgrading, according to the following scheme:

1) Sanitizer reads information, transforms it, and writes the resultant information in an object whose level is <u>equal</u> to that of the information read.

2) The security-level description of the object in which the resultant information resides is changed.

With the write-down method, sanitization and downgrading are done concurrently, with the sanitizing agent interposed between the original information and the object into which the resultant information is written. (See Figure 1 for a schematic comparison.)

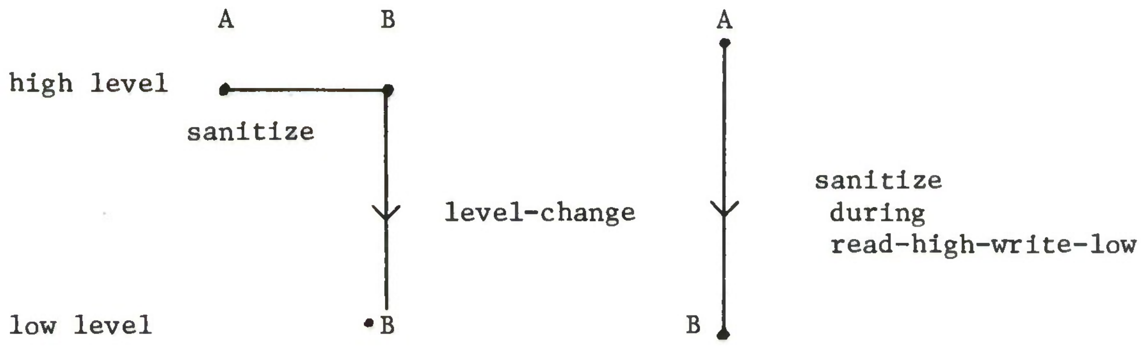


Figure 1. Temporal Interpretations of Sanitization

A second aspect of sanitization is the flexibility of the term "transformation". The computation of statistical measures of a population will provide an example of the range of meaning of this word. Suppose a file contains the names and weights of a certain group of men. One could transform the data in the file variously by:

computing the mean of the weights; or
grouping the weights into intervals and drawing a histogram; or
listing the men's names in ascending order of their weights; or
duplicating the file (the identity transformation).

That is, data can be blurred to widely varying degrees, and each degree of blurring represents a sanitization. Thus, a precise description of sanitization is highly application-dependent.

Internal and External Considerations

This last observation brings us to a discussion of external and internal considerations for downgrading. The Bell-LaPadula model for secure systems deals solely in terms of external controls. These include relations among entities in a system (for example, who has access to what) and attributes of an entity (such as a security classification). An example of an internal issue is the choice of a sanitizing transformation suited to the format and sensitivity of information to be transformed in a downgrading scheme.

If a system design that includes a downgrading capability is to be at all flexible and responsive to a user's needs, some portion of the downgrading procedures must depend upon internal requirements. On the other hand, external controls have the advantage that they can be designed and certified at an earlier stage of development than can internal controls, and before the system is tied to a particular application.

In view of these observations on external and internal considerations, the following questions are relevant:

1) For a given downgrading approach, what external controls can be devised?

2) How do such external controls affect the Bell-LaPadula model?

It is to these question that this section is addressed.

APPROACHES TO DOWNGRADING: MODEL IMPACT

In this section we will describe the forms that the two downgrading approaches introduced in the preceding subsection would take in the context of the Bell-LaPadula model of a secure computer system. Relative merits will be discussed in qualitative terms. The review of the Bell-LaPadula model included in the previous section was intended to set the context of that which follows.

Level-Change

The level-change approach would be grafted onto the model by the addition of a rule to ω_{iii}, the rule set of ESD-TR-73-278, Volume III. The level-change rule would have as arguments:

a subject name (the subject requesting the change);
an object name (the object whose level is to be changed);
$\left.\begin{array}{l}\text{a classification}\\ \text{a category}\end{array}\right|$ (the new security description of the object).

Given arguments of the proper form, the classification and category of the object referred to would be changed as requested if all appropriate criteria were satisfied. These criteria are:

1. The desired level of the object is not above its current level. This requirement, if satisfied, ensures that the level-change rule is security-preserving; this much is guaranteed by the Revised Basic Security Theorem of ESD-TR-73-278, Volume 1.

2. The desired level of the object is not below that of its directory. This requirement ensures the preservation of compatibility, which means that the security level of objects increases (or at least does not decrease) as one moves along the object hierarchy tree away from the root.

3 The requesting subject must have write-access to the directory of the object whose level is to be changed. This is necessary in order to record the change.

4. The desired level is not below the current level of any untrustworthy subject which has current write-access to the object whose level is to be lowered. This requirement prevents a *-property violation.

5. The subject has permission to lower the object's level.

6. The object contains no information which should not be accessible at the desired level of the object.

Criteria 5 (downgrading power) and 6 (object integrity) represent departures from the context of the model. They warrant separate discussion at this point.

Downgrading Power

Granting permission to lower the security level of an object implies reference to data which does not appear in the model. This data could take one (or a combination) of the following forms:

a simple downgrading-power flag for a subject;
a classification and category range over which an empowered subject may exercise its power;
downgrading empowerment on a per-subject, per-object basis.

Data structures supporting these forms of permission would be part of an external control mechanism, and therefore could be written into the model. The forms of permission listed above were not derived from any set of downgrading procedures in current use, but are means to serve solely as illustrations of the concept of empowerment permission. Consequently, it may be advisable to include in the kernel a facility for recognition of user-defined external controls of this nature.

Object Integrity

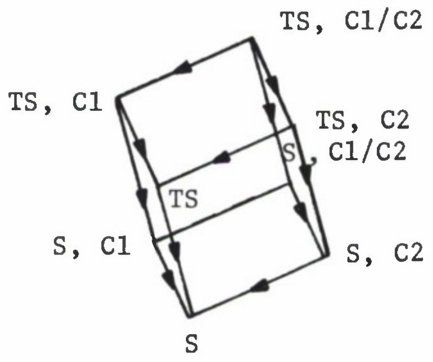
The requirement of object integrity (that the object whose level is to be lowered contains no information which should not be accessible at the desired level of the object) may strike one as being out of place in a discussion of modifications to the Bell-LaPadula model, since the use of internal characteristics for the purpose of meeting this requirement seems to be unavoidable. However, the burden of judgment that would rest upon internal mechanisms could be lightened by reference to historical records of external relationships among subjects and objects in the system. The value of such records is not confined within the level-change approach to downgrading, but extends to the write-down methods. Therefore, a more detailed discussion of this topic will be deferred.

Write-Down

A write-down capability is latent in the current model of a secure system, and can be exercised by a trustworthy subject.

The concept of a trustworthy subject was not introduced into the model for the purpose of downgrading, however. Certain subjects need to violate *-property in order for the system to progress -- for example, an "answering service" process that must communicate with users and initialize processes of various levels. Accordingly, members of a designated set of subjects would be allowed to have simultaneous read access to a high-level object and write access to a low-level object.*

*The *-property is written in the Bell-LaPadula model in such a way that information paths established by untrustworthy subjects are of a single level or lead upwards (with respect to the ordering of security levels). Since the collection of security levels is not a linearly ordered set, a subject that is exempted from the *-property may be able to "crossgrade" as well as downgrade. As an illustration, considering the following two-classification (secret (S) and top secret (TS)), two-category (C1 and C2) lattice:



An information path from a TS, C1 object to a TS, C2 object provides a crossgrading example; a TS, C1 to S, C2 path affords both crossgrading and downgrading opportunities; any path which can be traveled by following the arrows is a pure downgrading path.

The exemption would be justified by certifying that such subjects would not write down even though they were in a position to do so (hence the designation "trustworthy").

Although the exemption was modeled, details of its motivation were not. Consequently, an additional interpretation of trustworthiness is possible. In this interpretation, two types of trustworthy subjects are distinguished:

1) those subject that are exempted from obeying the *-property because they will not write-down.

2) those subjects that are exempted in order that they may write down.

Subjects of the second type are downgraders. Having identified the downgrading agents, which we will call formularies, we must decide how to control them. (The term "formulary" is taken from Hoffman [7]. A formulary, in Hoffman's usage, is a special-purpose collection of procedures that controls access to data at any desired level -- be it file, record, or even bit level -- and whose decisions that affect access may be based upon the user, the terminals being used, the time, and the content of the data themselves. The formulary stands between the user (or programs operating on his behalf) and the system programs that manipulate data items directly. The aspect of the above description of a formulary that is most relevant to our situation is content-dependence. (We will also use the word formulary to denote processes that execute such programs.) The simplest choice (from the modeler's point of view) is to write no further controls into the model, but to require the formulary writer to assume all responsibility for the definition of downgrading power and the safeguarding of object integrity. Structures which might aid the performance of the latter task are modeled later in this report.

Relative Merits of the Approaches

A positive feature of the level-change approach is the distance its use places between the different functional aspects of the model. Sanitization and downgrading are separated, and access controls related to these actions are applied separately.

A disadvantage of this approach is that dynamic alteration of security levels and maintenance of the *-property interfere with each other, so that attention to one of the two impairs the function of the other. Attempts to balance the two features are likely to clutter the model. For example, before the level of an object can be lowered, every subject with write access to that object must release its access in order to maintain the *-property.

Contrastingly, if the write-down approach is adopted, no alteration of the model is needed, as long as it is deemed acceptable to place a large portion of the responsibility for the protection of the system during downgrading upon the formulary writer. If it is decided to reserve a portion of this role for the kernel, data structures and record keeping routines would probably need to be added for support.

Summary

A variety of downgrading controls can be constructed, with a specific choice of a set of controls dependent upon:

1) the particular downgrading procedures to be used

2) the degree of flexibility desired in the system

3) the degree of complexity deemed acceptable in the primitives and data structures of the system.

Since (1) is very vague at this stage, it would be presumptuous to place values on (2) and (3). Therefore, it seems unwise to clutter the model with details which may later prove to be of little utility. Consequently, the write-down interpretation of the trustworthy subject feature of the model provides the favored approach to down-grading.

The task before us in implementing downgrading without compromising information consists of solving once more the problem to which the *-property was addressed, but without allowing *-property enforcement as a solution. What are needed are safeguards that are more sensitive to gradations of potential threats that is *-property enforcement. Some suggestions for structures that will support safe-guards will be introduced in the subsequent sections.

SECTION IV

FORMULARY CONTROLS

Downgrading is to be achieved within the secure computer system designed for the PDP-11/45 by the use of a class of processes called formularies. The following sections treat:

1. the definition of a formulary
2. controls inherent in the security kernel
3. user-formulary communication elements
4. user-formulary communication sequence
5. data integrity and correctness of formularies

THE DEFINITION OF A FORMULARY

A formulary is a trustworthy process, and as such it is exempt from *-property checks. The *-property is enforced in the mathematical model of computer security and therefore in the security kernel in order to prevent the construction of information paths from high-level objects. It is assumed that trustworthy processes will not abuse their ability to construct such a path, which implies that the code executed by trustworthy processes must be "certified". The bounds of certification will be explored later.

In the kernel, the special status of trustworthy processes appears in:

1) the get-write (getw) function--a trustworthy process may write into a segment of any level up to its own level.

2) the interprocess-communication-send (ipcsend) function-- a trustworthy process may send a message to a process of any level.

The above characteristics implement the defining property of trustworthy subjects.

CONTROLS INHERENT IN THE SECURITY KERNEL

The controls to which arbitrary processes are subjected by the security kernel are those of:

(1) security level comparisons;

(2) access control lists;

(3) preservation of compatibility;

(4) *-property enforcement.

The status of formularies as trustworthy processes exempts them from (4), but the other controls are still applicable. The effects of these other controls are:

(1) a process can write on segments of its own or lower security level;

(2) a process can only attempt to gain access to those segments to which it has been given specific access permission;

(3) a segment must have a security level greater than or equal to that of the directory in which it is described.

Let us posit a sequence of steps in which a formulary process is invoked by a user process. We will first examine the conditions imposed upon the elements of such a sequence.

USER-FORMULARY COMMUNICATION ELEMENTS

The elements of a user-formulary communications and downgrading sequence include:

A) a user process that wishes to have some information transformed to information of a different security level;

B) a segment (or segments) in which the information to be transformed resides;

C) a formulary process;

D) instructions (i.e., parameters) for the formulary, supplied by the user process;

E) a segment (or segments) in which the results of downgrading are written;

F) synchronization signals.

Recall that a security level has two components, a classification and a set of formal access categories. Classifications are ordered linearly (unclassified < confidential < secret < top secret), while the sets of formal access categories are partially ordered by set inclusion ($\subseteq$). This pair of orderings give rise to an ordering (symbolized by $\trianglelefteq$, as in [9]) of security levels given by

$$(\text{class}_1, \text{cat}_1) \trianglelefteq (\text{class}_2, \text{cat}_2) \text{ if and only if}$$

$$\text{class}_1 \leq \text{class}_2 \quad \text{and } \text{cat}_1 \subseteq \text{cat}_2.$$

Elements A-E listed above have security levels associated with them (A and C as processes; B, D, E as segments or segment-resident information), and (F) does also if such signals are transmitted through interprocess communications channels. We will refer to the security levels of A,...,E as L_u, L_o, L_f, L_p, L_r, respectively (the subscripts standing for _u_ser, _o_perand, _f_ormulary, _p_arameter, _r_esults).

Ignoring both model and design constraints and approaching the downgrading situation simple-mindedly, there are four access right relationships which must obtain:

a) The user can write the parameters

b) The formulary can read the parameters

c) The formulary has read access to the operand segment(s)

d) The formulary has write access to the result segment(s)

If security level constraints are considered, the above access right relationships require:

a') $L_u = L_p$ (untrustworthy processes write only at their own levels);

*b') $L_f \trianglerighteq L_p$;

c') $L_f \trianglerighteq L_o$;

d') $L_f \trianglerighteq L_r$ (since a formulary is trustworthy).

*$L_f \trianglerighteq L_p$ means $L_p \trianglelefteq L_f$. Also, $\triangleright$ means "$\trianglerighteq$ but not equal".

These relationships can be summarized graphically by[*]:

$$\begin{tikzcd}
 & & f \arrow[dl, "\underline{r}"'] \arrow[d, "\underline{r}"'] \arrow[dr, "\underline{w}"] & \\
u \arrow[r, "\underline{w}"] & p & o & r
\end{tikzcd}$$

Graph 1.

Although the term "downgrading" suggests $L_o \rhd L_r$, this relation is not included as a necessary one. Since the security levels form a nonlinear lattice in their ordering, "$L_o \rhd L_r$" would prevent the transformation of information to a <u>different</u> (as opposed to merely <u>lower</u>) level.

The relations described above can be reduced in number if parameters are transmitted via an interprocess communication mechanism instead of being written in a segment. If this route is taken, (a), (a'), (b), (b') above can be eliminated and replaced with

---

[*]Diagrams of this type will be used to illustrate security level/access right combinations. They are interpreted as follows: $\begin{array}{c}X\\ \downarrow\underline{x}\\ Y\end{array}$ means that X has access $\underline{x}$ to Y, and the security level of X is greater than or equal that of Y. The access right label ($\underline{x}$) may be omitted, and a horizontal connection indicates necessary equality of security levels.

(e) The user can use an ipc channel to the formulary, which implies

(e') $L_u \trianglelefteq L_f$,

which yields (together with (c') and (d') the diagram

$$\begin{tikzcd}
 & f \arrow[d, "\underline{r}"] & \\
u \arrow[ur, "ipc"] & o & r \arrow[ul, "\underline{w}"']
\end{tikzcd}$$

Graph 2.

Subsequent questions that must be considered include:

1) How does the formulary gain access to the operand segment?

2) How does the formulary gain access to the result segment?

3) How does the user relate to the operand segment?

4) How does the user relate to the result segment?

Security levels aside, in order for the formulary to access a given segment Y, process X (which may be the user, the formulary itself, the executive, or some other process) must add the formulary to segment Y's access control list. This requirement in turn implies that process X must write in the directory of segment Y. If process X is untrustworthy, then the levels of process X and the directory of segment Y must be equal.

It may not be reasonable to expect the formulary to have original control over the operand. If the user has control, then relative security levels and access rights are described by

$$\begin{tikzcd}
 & f \arrow[d, "\underline{r}"] \\
 & o \arrow[d, no head] \\
u \arrow[r] & do
\end{tikzcd}$$

(do = directory of operand)

Graph 3.

If X is not the user, then

$$\begin{tikzcd}
f \arrow[d, "\underline{r}"] & \\
0 \arrow[d, no head] & \\
do & X \arrow[l]
\end{tikzcd}$$

applies.

Graph 4.

If the result segment is specified by the user, then the user or some other process Ƶ will create and control the result segment:

$$\begin{tikzcd}
& f \arrow[d, "\underline{w}"] \\
& r \arrow[d, dash] \\
u & dr \arrow[l, "\underline{w}"']
\end{tikzcd}$$

Graph 5.

$$\begin{tikzcd}
r \arrow[d, "\underline{w}"] & \\
r \arrow[d, dash] & \\
dr & \text{Ƶ} \arrow[l, "\underline{w}"']
\end{tikzcd}$$

Graph 6.

If the formulary creates and controls the result segment, then we have

$$\begin{tikzcd}
f \arrow[d, "\underline{w}"'] \arrow[ddr, "\underline{w}"] & \\
r \arrow[dr, dash] & \\
& dr
\end{tikzcd}$$

Graph 7.

If (for the sake of illustration) the user is to be able to read the result segment, then Graphs 5, 6, and 7 become

$$\begin{tikzcd}
 & f \arrow[d, "\underline{w}"] & \\
u \arrow[r, "r"] \arrow[rr, bend right, "\underline{w}"'] & r \arrow[r, dash] & dr
\end{tikzcd}$$

Graph 5'*

$$\begin{tikzcd}
 & f \arrow[d, "\underline{w}"] & \\
u \arrow[r, "\underline{r}"] & r \arrow[d, dash] & \\
 & dr & z \arrow[l, "\underline{w}"']
\end{tikzcd}$$

Graph 6'

$$\begin{tikzcd}
 & f \arrow[d, dash, "\underline{w}"'] \arrow[rd, dash, "\underline{w}"] & \\
u \arrow[r, "\underline{r}"'] & r \arrow[r, dash] & dr
\end{tikzcd}$$

Graph 7'

Assuming that the disposition of

the parameter question (Graph 1 versus Graph 2),

the operand segment (Graph 3 versus Graph 4),

the result segment (Graphs 5, 6, 7),

and the readability of the result segment (5, 6, 7 versus 5', 6', 7') are independent matters, there are $2 \cdot 2 \cdot 3 \cdot 2 = 24$ composite security level/access right graphs that can be drawn to illustrate relations among the elements in a user-formulary communication sequence. Hopefully, a potential for modularity exists that would allow each of these combinations to be built within one system.

---

*$L_u \trianglerighteq L_r \trianglerighteq L_{dr} = L_u$ implies $L_u = L_r = L_{dr}$

USER-FORMULARY COMMUNICATION SEQUENCES

Regardless of the particular combination chosen from among those described in the previous section, three steps common to all user-formulary communication sequences can be identified:

1) A user invokes a formulary, specifying the operation to be performed and the operand;
2) The executive starts the appropriate formulary process, if necessary;
3) The formulary operates on the operand data and transforms it.

In this section, a sample user-formulary communication sequence is described which is based upon a particular (but arbitrary) choice of security level/access right combinations, and which includes steps 1-3 above. The sequence is written largely in terms of security kernel primitives. The lines of the sequence are numbered and labeled with letters U, E, or F according to whether the process executing that line is the user, executive, or formulary.

Before presenting the sample sequence, some words of explanation are in order on the extent to which a user is aware of the sequence. It is expected that a user-oriented formulary command language and associated language interpreter will be devised that will limit a user's view of the sequence to the invocation described in (1) above. The lines labeled U in what follows are executed on behalf of the user, but are hidden from him.

The assumption of the existence of a formulary command language leads to the problem of the certification of the command language interpreter. Similar questions should be pursued relative to the file management system, assuming that the user employs it to specify operands. Discussion of this issue is postponed until the next section.

The sequence that follows incorporates the following set of decisions:

1) Parameters are passed in a parameter segment

2) The user controls access to the operand segment (i.e., the user can write in the operand's directory)

3) The user controls access to the result segment

4) The user is able to read the result segment

These decisions are summarized in Graph 8, which is a composite of Graphs 1, 3, and 5'.

$$\begin{tikzcd}
 & & & f \arrow[ddll] \arrow[dr, "\underline{r}"'] \arrow[ddr, "\underline{w}"] & & \\
 & & & & o \arrow[dl, dash] & \\
dp \arrow[r, dash] & p & u \arrow[l, "\underline{w}"'] \arrow[r, "\underline{w}"] \arrow[ll, bend left, "\underline{w}"] \arrow[rr, bend right, "\underline{r}"'] \arrow[rrr, bend right=40, "\underline{w}"'] & do & r \arrow[r, dash] & dr
\end{tikzcd}$$

Graph 8.

Note that the system of two processes and six segments pictured in Graph 8 occupies at most three security levels, with the (untrustworthy) user process and five of the segments forced to have the same level. We emphasize that the relations pictured in Graph 8 are a consequence of decisions (1)-(4) above and controls inherent in the security kernel.

Finally, the sequence (with annotations to follow):

⋮

U1 *create* result segment
U2 *give* write access for result segment to formulary
U3 *create* parameter segment
U4 *give* self write access for parameter segment
U5 *give* read access for parameter segment to formulary
U6 *getw* and *enable* parameter segment
U7 write in parameter segment:
  U7a operand segment identification and operand locations
  U7b result segment identification
  U7c other parameters
U8 *disable* and *release* parameter segment access
U9 *give* read access for operand segment to formulary
U10 *ipcsend* message to formulary: parameter segment identification
U11 *ipcsend* message to executive: start formulary, if necessary
U12 puts self to sleep.
E1 *ipcrcv* message from user
E2 *startp* formulary, if necessary
F1 *ipcrcv* message from user
F2 *getr* and enable access to parameter segment
F3 evaluate segment integrity and decide whether to proceed
F4 *getr* and *enable* access to operand segment
F5 *getw* and *enable* access to result segment

F6 operate: operand $\longrightarrow$ results

F7 *disable* and *release* all accesses to parameter, operand, and result segments

F8 *ipcsend* message to user: done

F9 *stopp*

U12 *ipcrcv* message from formulary

U13 *getr* and *enable* result segment

.
.
.
.

Remarks and annotations

It must be emphasized that this sequence is meant to serve as a sample of a class of sequences that could be described. As such, it is subject to revision or discarding.

U7c, F6. These steps are completely application-dependent.

U8, F7. The principle which is being applied is that a process should release access to segments when those segments are no longer needed by that process.

U9. This requires the addition of $\overset{f}{\underset{u}{\uparrow}}$ ipc to Graph 8.

F3. This step poses the most difficult problems (in terms of security) for the user-formulary communications sequence. The topic will be discussed more fully in the following section.

F4, F5. The difficulties alluded to above aside, minimal guarantee of integrity of the segments involved in the operations of the formulary can be effected during the operations of the formulary. This guarantee takes the form of a requirement that no other process can write a segment while a formulary process is reading that segment.

The enforcement of this requirement could be accomplished in at least three ways. First, one could make some modifications to the security kernel, either to <u>enable</u> or to <u>getr</u>/<u>getw</u>:

1) formulary cannot gain access to a segment that is being written by another process.

2) A process cannot gain write access to a segment that is being accessed by a formulary.

The application of the above conditions depends upon an inspection of the connected process list of an active segment table entry. Alternatively, a formulary could remove other processes from access control lists, employing its write-down capability as needed.

A second (and simpler) way would be for the formulary process to create a blank segment, copy the operand segment into the new segment, and operate on the copy.

F5, F7. Recognition of trustworthy processes is incorporated in the getw and ipcsend function.

F6. This includes a determination that the result segment of a level suitable for the results derived by the formulary.

OBJECT INTEGRITY AND CORRECTNESS OF FORMULARIES

As stated in section 1, it is assumed that trustworthy processes will not abuse their ability to construct paths from segments of high security levels to segments of a lower level. For this assumption to be valid, formularies need to possess two characteristics that are to some extent independent of each other:

(a) The programs executed by formularies must be known to be correct in their operations;

(b) The formulary must be able to evaluate the threats to which operand data has been exposed.

An example that will distinguish between these two points relies upon the concept of a Trojan horse program. Suppose that intelligence sources (agents, remote sensors) whose physical characteristics and observations are classified top secret gather data and transmit it to an intelligence data base. Formularies act upon the TS data and derive information that will be used by tactical personnel with secret clearances. (This sequence of events is schematized in Figure 2.) One of the steps in the operation of the formulary is

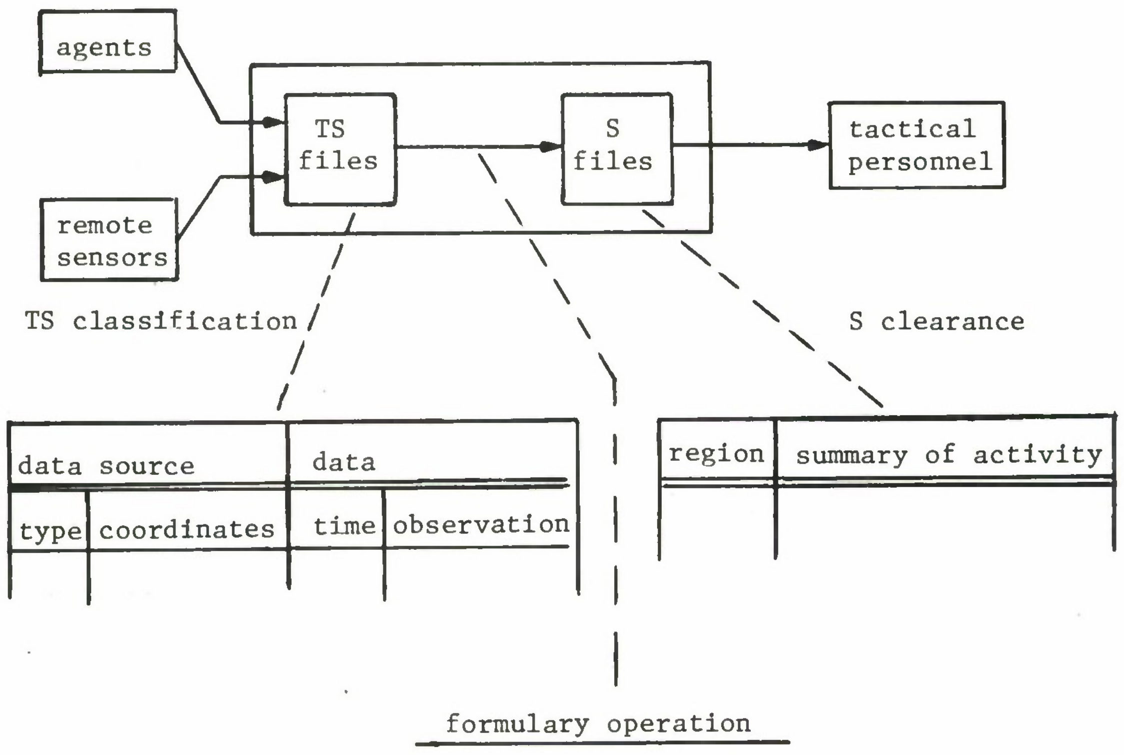


Figure 2. An Operational Configuration with a Downgrading Scheme

the specification and fetching of the TS data upon which the operations are to be performed. Suppose this step is performed with the assistance of a general purpose file management system that contains a section which can masquerade as a formulary when it recognizes TS data. That is, the file management system is a Trojan horse, and the correctness of the formulary's transformation operations cannot prevent the file management system from taking advantage of the trustworthiness (i.e., write-down capability) of the formulary process on whose behalf it executes. Therefore, the presence of characteristic (a) coupled with the absence of characteristic (b) abets a situation which is undesirable from the standpoint of security.

Characteristic (a) appears to be entirely application-dependent. Consequently, we will assume that the writers of formulary programs will be able to guarantee its presence, and we will address the problems posed by characteristic (b).

Directly posed, the question which should be abstracted from the above example is as follows:

Are the contents of a given segment such that it is safe to apply a given sanitizing algorithm to them? Phrased in this form, the question demands an answer which relates to the contents of the segment. Consequently, in order to solve the problem as stated, a formulary process must interpose (in time) between its invocation and the application of its transformation programs either

(a) smart programs that can assess the suitability of the contents of a segment for downgrading, or

(b) a human with such capabilities.

Resorting to controls such as these both postpones confrontation with the problem and shifts the burden of its solution to the formulary writer. In order to search for an earlier resolution in an absolute (rather than content-relative) vein, a less direct question is in order: What relationship with other subjects and objects (i.e., process and segments) has a given segment enjoyed prior to its use as an operand segment by a formulary?

This question, insofar as it can be answered in terms of access rights and combinations of access rights, is independent of content. Thus, an attempt can be made to answer it in the presently available context. We begin by reviewing the components of that context.

The entities that must be related to each other include processes and segments, programs and data, and the available modes of protection. Security levels aside, we discern at present two kinds of processes (trustworthy and untrustworthy) and two kinds of programs (certified and uncertified). Processes and programs relate to each other through the following observations:

Processes seek to gain access to segments.

Program code resides in segments.

Trustworthy processes execute only certified programs.

Note that the last observation does not exclude the possibility that untrustworthy processes may execute certified programs.

Since all operations upon segments are effected by the execution of some program code, and since program code resides in segments, the integrity of a segment can be defined in terms of the segments that have "touched" it. The access privileges enjoyed by processes with respect to a given pair of segments determine whether one segment has touched the other. The access protections that can be afforded a segment by the 11/45 hardware (without the use of D registers) are for no access, read/execute access, and write/read/execute access; execute-only is indistinguishable from read/execute. Hence, segments containing pure program code cannot be treated differently from read-only data segments.

If a process has read access to segment A and write access to segment B, then A is capable of touching (i.e., affecting) B. Since this attempt at integrity analysis purposely avoids the semantics of a computation and deals only in access rights, no distinction will be drawn between <u>possessing</u> a capability and <u>exercising</u> a capability. Thus, we will say that A touches B if A is capable (as described above) of touching B.

The aforementioned relationship is the most direct touching possible between distinct segments. It can be used as a building block either transitively (if A touches B, and B touches C, then A touches C) or temporally (if a process <u>had</u> read-access to A and <u>has</u> write-access to B, then A touches B) to construct more complex relationships among segments.*

However, complex the relations may be, they admit of a summary as a one-bit "history" per segment. Namely, if any uncertified segment has touched a segment A (however touching might be defined), then the integrity of A may have been compromised. Indeed, given the convention discussed above that capability = actuality, we assume that A has been compromised.

This criterion for compromise may have a very restrictive effect on the use of formularies. For, any process is likely to access a

---

*This subject is elaborated upon in terms of the Bell-LaPadula model in the next section.

low security level uncertified system program (such as a compiler or file management system) and a given segment concurrently, thereby rendering the segment unsafe for formularies to operate upon without the intervention of a human or automatic content analyzer. Thus, the price paid for the integrity of the system may be non-functionalism.

To relieve this condition, several alternatives are available. One course of action would be to certify all frequently used system programs. Given the large size of such programs, this seems an onerous task, and one that should be avoided.

Another way to alleviate the problem would be to label as _non-malicious_ those untrustworthy processes that are "known" to be free of evil intentions, and to allow only such processes to invoke formularies or to touch segments that will be downgraded. This gradation of untrustworthy processes does not by itself speak to the problem, since the problem arises from potential properties of segments, not of processes.

In view of this, it might be more fruitful to introduce a concept of non-maliciousness for segments. The term "non-malicious" will be discarded as being misleading, for a segment which is not non-malicious is not necessarily the tool of a malicious agent; rather, it is not _known_ to be non-malicious. A better adjective than "non-malicious" is "safe". If a segment is marked as safe, then its touch will not impair the segments it touches with regard to the application of formularies.

The justification for attaching the safe label to a given segment is a matter separate from this operational definition of safety. In this regard, the concept of "safe object" is analogous to that of "trustworthy subject", insofar as we define the latter notion in terms of the exceptional properties a trustworthy subject has, without saying why it has those properties. A segment might merit the appellation "safe" if it were

(a) certified; or

(b) written by trusted personnel and touched only by other safe segments.

In a refinement of (b) which has been suggested (and which is akin to the sabotage model mentioned in Section II), a segment would be safe up to a particular security level. That is, a safe segment could touch segments with a security level less than or equal to the designated level without impairing the applicability of formularies to those segments. The safe label refers to a segment and _not_ to the program code it might contain. Thus, a malicious user might be able to copy and alter a system program, but he could not masquerade it as the original, safe program unless he could assign to the segment(s) in which the copy resides the level of safety of the original segment(s).

The usefulness of this last refinement is hampered by the determination of the limiting security level. If the level that defines the limit of safety of a segment is the level of the segment, then no segment may touch one of a higher level without impairing the higher level segment; that is, after all, the original problem which motivated the current discussion. Thus, in order for the notion of a limit of safety to be useful, the limit of a segment should be greater than the level of that segment. A reasonable limit to assign to a segment would be the level of the process that creates it. This assignment is reasonable in terms of the Bell-LaPadula model with its concepts of current and maximum levels of subjects. However, the current security kernel design recognizes only current levels for processes, and constrains a process to have a level less than or equal that of any segment it creates. If the assignment of a limit of safety of a segment were made in the manner suggested above, the limit would be bounded above by the level of the segment; such an arrangement is counterproductive. Thus, if the idea of "limit of safety" is to be useful, some other procedure for the assignment of the limit must be devised.

The concept of levels of safety permits the problem of the uncertified file system mentioned earlier to be defined away. Namely, if it is assumed that the segments comprising the file system are safe to the level of every process in the system, then the problem disappears (as long as the file system itself is untouchable).

Once it has been decided to wave a magic wand over one important system program in this manner, the extension of the technique to other programs (such as compilers) can be considered.

In summary, the maintenance of the integrity of a segment can be accomplished by

(a) controls based upon access rights and security level comparisons and

(b) automatic or human analysis of the contents of an operand segment at the time a formulary is invoked.

SECTION V

MODELLING DATA PATHS

INTRODUCTION

In this section we will discuss the detection of threats to data that formularies use as operands. The approach taken starts at the level of the mathematical model and consists of the construction of data paths (audit trails) from information contained in the record of current access.

The possible use of data paths will not be closely specified here, since such use is highly application-dependent. Rather, several levels of data paths that can be inferred from the record of current access will be described. Moreover, at each level of inference, differing degrees of detail in the data path description will be available.

TWO PRINCIPLES OF PATH CONSTRUCTION

The data path model that will be developed here relies upon two principles. They are:

1) A comprehensive record of data paths can be derived from observations of the record of current access.

2) The relations essential to a description of data paths are those among objects. Relations that include subjects only add detail to path descriptions.

The first of these principles is explained and justified easily. The opportunity to move information arises only from combinations of access relations. Therefore, analysis of a complete account of access relations will yield all information paths. For the moment we will ignore the difficult problem of the derivation of all paths.

The most straightforward example of a combination of access relations that results in an information path consists of a process that simultaneously enjoys read access to one segment and write access to another. This most obvious and direct situation does not exhaust all arrangements that are of interest. Simultaneity of the access relations or reliance upon a _single_ process may be omitted from the above example. A path may still exist between two segments if access relations exist during the progress of the system that suffice to connect them.

To reemphasize the first principle, regardless of the complexity of relations that contribute to a data path, the path can be inferred solely from a record of access relations. Note, however, that the completeness of the data path record is directly dependent upon the completeness of the record of access relations.

The second principle, that path descriptions should be object-based, derives from the observation that information resides in objects. An object-object link is the minimal necessary evidence of an information path. The identity of the subject or subjects that effect such a link are embellishments that are probably useful additions to the path description, but can be done without at the lowest level of the description. The same comment applies to the access modes that are used in the construction of a path.

IDENTIFICATION OF PATH TYPES

The record of current access (the b of the model) is a collection of entries of the form (subject, object, access mode) that conveys the information that a particular subject currently enjoys a specified mode of access to a specified object. If the record contains the pair of entries (S_i, 0_j, _read_) and (S_i, 0_k, _write_), a path from 0_j to 0_k may be inferred; this arrangement is a formal version of the "most straightforward example" of the previous subsection (Figure 3).

It is worthwhile to describe a few variations on this theme. If there are entries (S_m, 0_k, _read_) and (S_m, 0_n, _write_) in addition to the two entries given above, then there is a path from 0_j to 0_n (Figure 4). If the third or fourth entry is recorded after the other entries have been made, the path from 0_j to 0_n exists (Figure 5. Further, if the first or second entry is removed, the path still exists as a feature of interest (Figure 6).

A more elaborate example is indicated by Figure 7. (Subjects are suppressed in Figure 7, in accordance with the principle of object-based paths). Supose that at time t there are paths from object A to object B and from B to object C. At time t+1 the link between B and C is broken; at time t+2, paths from D to A and from C to E are established. Beyond time t+2, information which at one time lay in A may come to reside in E, so that a path from A to E should be noted. However, information cannot move from D to c, so that path from D to C need not be noted.

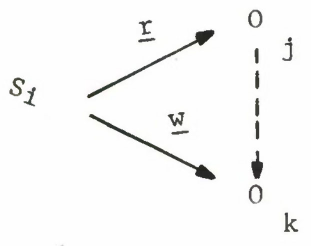


Figure 3. Direct Path

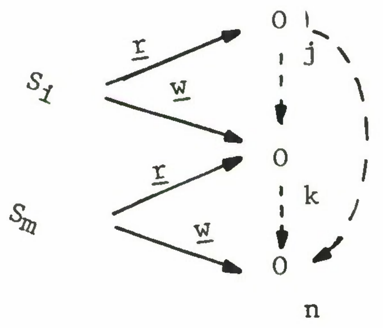


Figure 4. Indirect Path

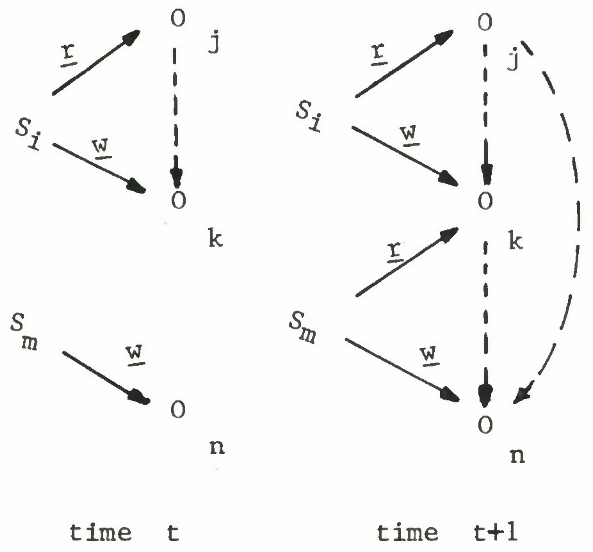


Figure 5. Evolving Indirect Path (Entries Intact)

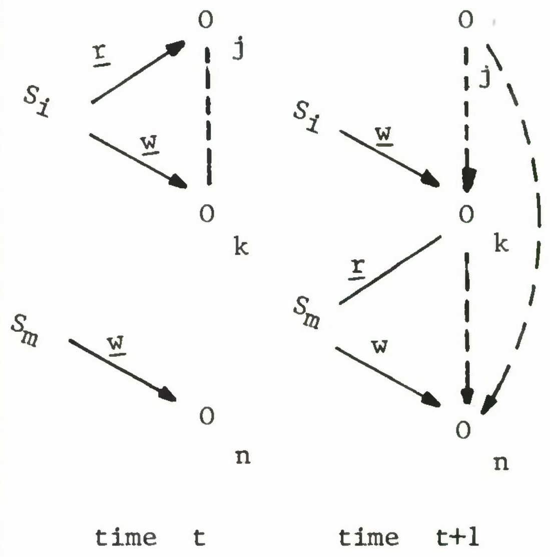


Figure 6. Evolving Indirect Path (Entry Removed)

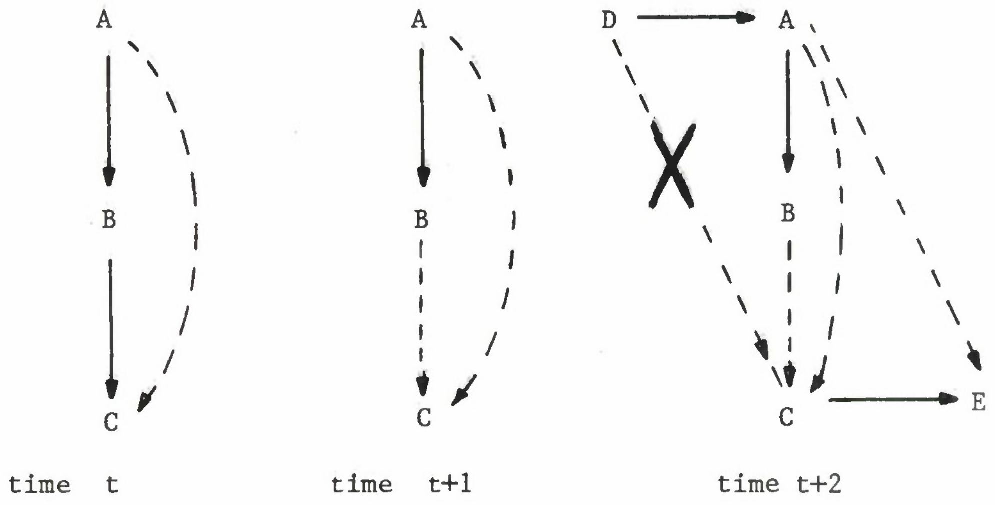


Figure 7. Evolving Nonpath

We can characterize the types of paths described above concisely:

1) direct (Figure 3)

2) indirect (Figure 4)

3) time-dependent (Figures 5, 6, 7)

Figure 7 also indicates an unuseable path, which we will call a

4) nonpath

FORMALIZING PATHS

We begin by refining the principle of path derivability, as follows:

Assume that every data path can be built by concatenation of direct paths; and that all direct paths can be inferred from the record of current access.

Thus, the manner in which direct paths are inferred from the record of current access is crucial to data path construction. Postulate the existence of a procedure that extracts direct paths from a record-of-current-access-like set. Formally, such a procedure corresponds to a function.

$$d_B\colon \mathcal{O} \times \mathcal{O} \longrightarrow \mathcal{P}(\mathcal{S} \times \mathcal{A} \times \mathcal{A}, \text{ where } B \subseteq \mathcal{S} \times \mathcal{O} \times \mathcal{A}.$$

That is, given a pair of objects O_j, O_k, a search is made through B, which is a collection of entries of the form (subject, object, access mode), for pairs of entries

$$(S_i, O_j, x_1), \quad (S_i, O_k, x_2)$$

which indicate that there is a direct path from O_j to O_k. When such a pair is found, the triple (S_i, x_1, x_2) is used to represent the link between O_j and O_k. $d_B(O_j, O_k)$ is the collection of all such triples.

If x_1 = <u>read</u>, x_2 = <u>write</u>, then (S_i, x_1, x_2) represents the arrangement of Figure 3. Values of x_1 = <u>write</u>, x_2 = <u>read</u>, or x_1 = x_2 = <u>read</u> presumably would not occur in the direct path record, for these combinations do not promote the passing of information.

The existence of a direct path from O_j to O_k turns on whether $d_B\ (O_j, O_k)$ contains any entries or is an empty set. If it is desirable to know <u>how</u> a path was built, or <u>who</u> built it, the value of $d_B\ (O_j, O_k)$ can be inspected in detail.

The semantics of information passage are most important at the level of direct paths. Beyond that level, construction of data paths is largely a combinatorial and set theoretical problem. For an unchanging set B, a path is formalized as a sequence $0_{j_1}, 0_{j_2}, \ldots, 0_{j_k}$ of objects such that $d_B\ (0_{j_p}, 0_{j_{p+1}}) \neq$ empty set, p+1,...,k=1; that is, there are direct paths between consecutive objects in the sequence. The collection of all paths derivable from B is denoted by P(B).

Once again, several levels of detail are available in the use of data paths. A given implementation of the data path model might make use of

the existence of a path from 0_{j_1} to 0_{j_k};

the full list of objects comprising the path;

the subjects that effect the direct paths along the path from 0_{j_1} to 0_{j_k};

or the access modes available to form the direct paths between consecutive objects.

As a system for which data paths are to be constructed evolves, the record of current access changes. Suppose (following Bell and LaPadula) that the sequence of records of current access of a system is denoted by $b^{(t)}$, t=0,1,2,... A likely candidate for a dynamic record of data paths can be constructed by aggregating all records of current access and using the aggregate as the "B" needed for the construction of data paths. Formally, if we denote the data path record to time t by $P^{(t)}$, then

$$P^{(t)} = P(\bigcup_{i=o}^{t} b^{(i)})$$

Again, several levels of detail are possible in the use of this model.

This model not only covers the useable paths of Figures 5, 6, and 7, but it also includes the unuseable path of Figure 7. In this sense, therefore, the proposed model finds too many paths. However, no attempt will be made at this point to refine the model to eliminate such paths. The reason for this decision is that Figure 7 illustrates the simplest sort of unuseable path. The system of objects pictured is small, isolated, and contains no loops; the recognition of unuseable paths in general is a difficult problem whose solution would exact a high cost in complexity.

USING THE DATA PATH MODEL

Assume that a security kernel that is based upon the Bell-Lapadula model has been designed. The first step in the design of a data path record is the determination of those combinations of access rights that constitute direct paths. We emphasize once more that the record of current access must be complete with regard to the collection of objects. For example, the Active Segment Table of the PDP-11/45 security kernel is only <u>part</u> of the record of current access, since it (the AST) only describes which processes have direct read or write access to one type of object (namely, segments).

The next step is the development of an algorithm that will extract direct paths from the data path base (the set that has been called "B" to this point). The data path base itself must be updated with each change in the record of current access. The data path base will be a larger structure than the record of current access (in terms of the number of entries), and it will be searched often. Therefore, maintenance of the data path base includes an organization (and reorganization) which will facilitate searches.

Assuming that the data path base is faithfully maintained, the data path record need only be recomputed or updated when it is needed, since the accuracy of the data path record is dependent only upon an accurate data path base.

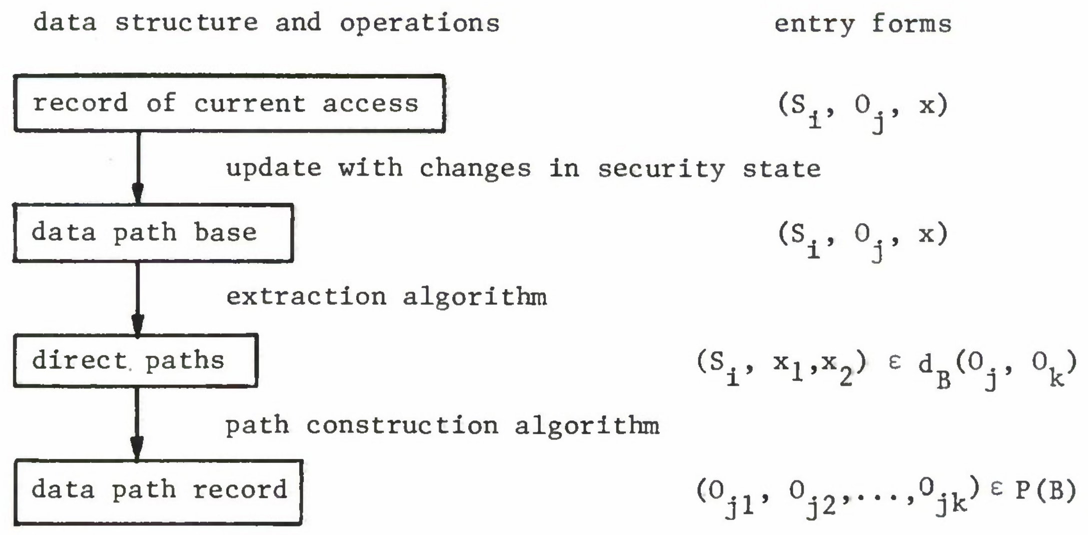

data structure and operations
entry forms
record of current access
(S_i, O_j, x)
update with changes in security state
data path base
(S_i, O_j, x)
extraction algorithm
direct paths
$(S_i, x_1, x_2) \varepsilon d_B(O_j, O_k)$
path construction algorithm
data path record
$(O_{j1}, O_{j2}, \ldots, O_{jk}) \varepsilon P(B)$

Figure 8. Schematic Flow of Data Path Model

CONCLUSION

Controlled downgrading by formularies requires that data is transformed correctly, and that transformations are only performed upon data which has not been mishandled prior to formulary invocation. The first requirement is application-dependent, but the second requirement can be partially met by reference to a record of data paths that, in its abstract form, is phrased in terms of the Bell-LaPadula model of a secure system.

LIST OF REFERENCES

1. S. B. Lipner, "Computer Security Research and Development Requirements," MITRE Corporation, MTP-142, February 1973.

2. D. E. Bell and L. LaPadula, "Secure Computer Systems," ESD-TR-73-278, Volumes I, II, III, November 1973 - April 1974.

3. W. L. Schiller, "Design of a Security Kernel for the PDP-11/45," ESD-TR-73-294, December 1973.

4. PDP-11/45 Processor Manual, Digital Equipment Corporation, 1973.

5. G. S. Graham and P. J. Denning, "Protection Principles and Practice," AFIPS Conference Proceedings, Volume 44, SJCC (1972).

6. AFR-205-1, "Safeguarding Classified Information," Department of the Air Force, January 1968.

7. L. J. Hoffman, "The Formulary Model for Access Control and Privacy in Computer Systems," dissertation, Stanford Linear Accelerator Center, May 1970.

8. J. C. C. White, "Design of a Secure File Management System," ESD-TR-75-57, April 1975.

9. Stanley R. Ames, Jr., "File Attributes and Their Relationship to Computer Security," M. S. dissertation, Case Western Reserve University, June 1974.